A

# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 09/671,770 | 09/28/2000 | Geoffrey Owen Blandy | AUS9-2000-0570-US1 | 7435 |

7590 11/24/2003

Duke W Yee
Carstens Yee & Cahoon LLP
P O Box 802334
Dallas, TX 75380

| EXAMINER | |
|---|---|
| ALI, SYED J | |
| **ART UNIT** | **PAPER NUMBER** |
| 2127 | 2 |

DATE MAILED: 11/24/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

| *Office Action Summary* | Application No. | Applicant(s) |
|---|---|---|
| | 09/671,770 | BLANDY ET AL. |
| | Examiner | Art Unit |
| | Syed J Ali | 2127 |

***-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --***

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☒ Responsive to communication(s) filed on 28 September 2000.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) 1-30 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-30 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. §§ 119 and 120**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

13)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application) since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

a) ☐ The translation of the foreign language provisional application has been received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

**Attachment(s)**

| | |
|---|---|
| 1) ☒ Notice of References Cited (PTO-892) | 4) ☐ Interview Summary (PTO-413) Paper No(s). _____ . |
| 2) ☒ Notice of Draftsperson's Patent Drawing Review (PTO-948) | 5) ☐ Notice of Informal Patent Application (PTO-152) |
| 3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____ . | 6) ☐ Other: . |

## DETAILED ACTION

### *Claim Rejections - 35 USC § 102*

1. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

2. Claims 1-7, 11-17, and 21-27 are rejected under 35 U.S.C. 102(e) as being anticipate by Lueh et al. (USPN 6,158,048) (hereinafter Lueh).

As per claim 1, Lueh discloses a method of calling a portion of computer code in a multithreaded environment, comprising:

receiving a call to the portion of computer code (col. 6 line 23 - col. 7 line 14, "Initially, at step 410, code selector takes a bytecode or expression from the code stream");

determining if the portion of computer code is currently being compiled (col. 6 line 23 - col. 7 line 14, "Before selecting code for the bytecode, the selector looks ahead in the stream to see whether the expression starting at this bytecode matches one already associated with a scratch register"); and

redirecting the call to an interpreter, if the portion of computer code is currently being compiled (col. 6 line 23 - col. 7 line 14, "If the expression comparison at step 430 indicates that the expression are syntactically identical, then the code selector pushes the contents of the register onto the stack at step 450", wherein if the call matches code that is currently being compiled, the expression is redirected such that the compiled expression is used as the result of the subsequent code).

As per claim 2, Lueh discloses the method of claim 1, wherein the portion of computer code is a Java method (col. 5 line 43 - col. 6 line 22, "The system 300 also includes a Java Virtual implementation 330 for running Java class files 360").

As per claim 3, Lueh discloses the method of claim 1, wherein redirecting the call to an interpreter includes redirecting the call to a Java Virtual Machine Interpreter such that the portion of computer code is interpreted by the Java Virtual Machine Interpreter in response to receiving the call to the portion of computer code (col. 5 line 43 - col. 6 line 22, "the Java Virtual Machine 330 may utilize a Java Interpreter 332 for interpreting Java class files 360 or a Java 'Just-In-Time [JIT} computer 334 to generate compiled Java code", wherein the elimination of common subexpressions is done by redirecting a particular portion of code to the compiled code generated by the JIT compiler).

As per claim 4, Lueh discloses the method of claim 1, wherein determining if the portion of computer code is currently being compiled includes determining a setting of a flag in a control block of the portion of computer code (col. 6 line 23 - col. 7 line 14, "If no stored expressions are available, the code selector selects an instruction sequence for the bytecode at step 460. Then the expression value is stored in a register at step 462 and the expression tag of the register containing the result of the instruction sequence is updated", wherein the expression tags are evaluated at a later point when a match is sought for the current expression being evaluated).

As per claim 5, Lueh discloses the method of claim 1, wherein the step of redirecting the call is performed in response to a Just-In-Time [JIT] invoker field, in a control block of the portion of computer code, pointing to a JIT to Java Virtual Machine [JVM] routine (col. 5 line 43 - col. 6 line 22, "the Java Virtual Machine 330 may utilize a Java Interpreter 332 for interpreting Java class files 360 or a Java 'Just-In-Time [JIT] computer 334 to generate compiled Java code", wherein upon locating a match for a subsequent expression, the result of the current bytecode being compiled is used as the result for the subsequent expression. The program flow is disclosed within a system that utilizes a Java Virtual Machine, wherein the compiled code is generated from a Just-In-Time compiler).

As per claim 6, Lueh discloses the method of claim 1, further comprising:

determining if compilation of the portion of computer code has ended (col. 6 line 23 - col. 7 line 14, wherein the expression being compiled is compiled to completion, and then a search is done to see if subsequent calls within the code stream make identical system calls. In this sense, the redirection of the expression to the compiled code of the 'current' expression is done after the expression has been evaluated, and thus compilation has ended); and

redirecting the call to a compiled version of the portion of computer code if the compilation of the portion of computer code has ended (col. 6 line 23 - col. 7 line 14, "If the expression comparison at step 430 indicates that the expression are syntactically identical, then the code selector pushes the contents of the register onto the stack at step 450", wherein the values pushed onto the stack come from the expression that is 'currently' undergoing compilation, i.e., once the 'current' expression has been evaluated and the registers updated, the JIT compiler looks ahead in the code stream for other expressions with an identical system call).

As per claim 7, Lueh discloses the method of claim 6, wherein redirecting the call to a compiled version of the portion of computer code is performed in response to setting a Just-In-Time [JIT] invoker field, in a control block of the portion of computer code, to point to the compiled version of the portion of computer code (col. 5 line 43 - col. 6 line 22, "the Java Virtual Machine 330 may utilize a Java Interpreter 332 for interpreting Java class files 360 or a Java 'Just-In-Time [JIT] computer 334 to generate compiled Java code", wherein the redirection of subsequent expressions to be evaluated is done in response to finding a match for the expression, and does not depend on whether the expression has completed compilation or not. Rather, the execution stack is modified for the subsequent expressions as matches are found. Thus, in order for the results to be used, the compilation must be complete).

As per claims 11-17, Lueh discloses an apparatus for implementing the method of claims 1-7, respectively. Specifically, Lueh is disclosed as a method of optimizing Java bytecodes as they are compiled. This is done through the use of a JIT compiler that runs on a Java Virtual Machine, which in turn runs on any type of computer system that supports Java. Therefore, the host computer, which runs the JVM, is an apparatus on which the method of claims 1-7, respectively, may be implemented.

As per claims 21-27, Lueh discloses a computer program product in a computer readable medium for implementing the method of claims 1-7, respectively. Specifically, Lueh is disclosed as a method of optimizing Java bytecodes as they are compiled. This is done through the use of a JIT compiler that runs on a Java Virtual Machine, which in turn runs on any type of computer system that supports Java. The entire disclosure of Lueh is related to optimization of computer code, which also must be implemented as computer code for the machine to be able to properly interpret the method. Therefore, the discussion of claims 1-7 form the basis for rejection of the present claims as well.

## *Claim Rejections - 35 USC § 103*

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4. Claims 8, 18, and 28 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lueh in view of Levine et al. (USPN 6,311,325) (hereinafter Levine).

As per claim 8, Levine discloses the following limitations not shown by Lueh, specifically the method of claim 1, wherein the portion of computer code is a Java method having an associated method block and wherein the steps of determining if the portion of computer code is currently being compiled and redirecting the call are performed based on information stored in fields of the method block (col. 7 lines 43-52, "a determination is made as to whether a compiled address is present [step 812]. This determination is made by examining the flag obtained in [step 810]. If a compiled address is present, the compiled address is retrieved from the JITs table", wherein the flag referred to indicates whether or not a particular method has been compiled)

It would have been obvious to one of ordinary skill in the art to combine Lueh with Levine since the use of a flag would permit the JVM to make a quick and easy check to determine whether or not a method has been compiled or is undergoing compilation. Specifically, if a table were updated to indicate that a method has been JITed, any subsequent calls to that method would immediately know that a compiled version is available or soon will be. Since JIT compilers are sought to be lightweight, since compilation is part of the runtime execution, reducing the burden such that a check can be done by comparing a single bit is advantageous to checking each bytecode sequence for a match, as Lueh does. Therefore, the combination of Lueh and Levine provides a way of eliminating redundant code sequences, while also reducing the time required to perform sub searches of code, which could be extremely costly in cases of longer code streams.

As per claim 18, Lueh discloses an apparatus for implementing the method of claim 8. Specifically, Lueh is disclosed as a method of optimizing Java bytecodes as they are compiled. This is done through the use of a JIT compiler that runs on a Java Virtual Machine, which in turn runs on any type of computer system that supports Java.

Therefore, the host computer, which runs the JVM, is an apparatus on which the method of claim 8 may be implemented.

As per claims 28, Lueh discloses a computer program product in a computer readable medium for implementing the method of claim 8. Specifically, Lueh is disclosed as a method of optimizing Java bytecodes as they are compiled. This is done through the use of a JIT compiler that runs on a Java Virtual Machine, which in turn runs on any type of computer system that supports Java. The entire disclosure of Lueh is related to optimization of computer code, which also must be implemented as computer code for the machine to be able to properly interpret the method. Therefore, the discussion of claim 8 forms the basis for rejection of the present claim as well.

5. Claims 9-10, 19-20, and 29-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lueh in view of Levine in view of Beadle et al. (USPN 6,324,687) (hereinafter Beadle).

As per claim 9, Beadle discloses the following limitations not shown by the modified Lueh, specifically the method of claim 8, wherein the method block includes a field that includes a pointer that points to a Java Virtual Machine [JVM] interpreter before a Just-In-Time [JIT] compiler is loaded, points to a JIT compiler routine CompileThisMethod when the JIT compiler is loaded, and points to a routine which calls a compiled version of the method once the method is compiled by the JIT compiler (Fig. 5, wherein each class method has a flag that indicates whether or not it is to be 'JITed'.

In the case where there is no JIT compiler loaded, the JVM inherently must interpret the method without the JIT compiler. On the other hand, the CompileClass( ) method corresponds to the CompileThisMethod function in that it is how the JIT compiles the class or method).

It would have been obvious to one of ordinary skill in the art to combine the modified Lueh with Beadle since the compilation method of Beadle further allows the JVM to specialize the JIT such that only particular methods are JITed, or reduces the need for excessive compilation of redundant methods. Specifically making reference to Lueh, if a match of a bytecode expression is found for a method that is undergoing compilation, the flag in Fig. 6 of Beadle could be modified for that method to "No JIT", and rather pushing onto the stack the results of the compiled expression. The combination therein provides a sound data structure that not only allows a user to specify which methods should be compiled, but also has a provision that may prevent redundant compilation of methods at runtime.

As per claim 10, Beadle discloses the following limitations not shown by the modified Lueh, specifically the method of claim 8, wherein the method block includes a field having a pointer that points to a Just-In-Time [JIT] compiler routine CompileThisMethod when the JIT compiler is loaded and points to a compiled version of the method when compilation of the method by the JIT compiler is complete (Fig. 5, wherein each class method has a flag that indicates whether or not it is to be 'JITed'. In the case where there is no JIT compiler loaded, the JVM inherently must interpret the method without the JIT compiler. On the other hand, the CompileClass( ) method corresponds to the CompileThisMethod function in that it is how the JIT compiles the class or method).

It would have been obvious to one of ordinary skill in the art to combine the modified Lueh with Beadle for reasons discussed above in reference to claim 9.

As per claims 19-20, Lueh discloses an apparatus for implementing the method of claims 9-10, respectively. Specifically, Lueh is disclosed as a method of optimizing Java bytecodes as they are compiled. This is done through the use of a JIT compiler that runs on a Java Virtual Machine, which in turn runs on any type of computer system that supports Java. Therefore, the host computer, which runs the JVM, is an apparatus on which the method of claims 9-10, respectively, may be implemented.

As per claims 29-30, Lueh discloses a computer program product in a computer readable medium for implementing the method of claims 9-10, respectively. Specifically, Lueh is disclosed as a method of optimizing Java bytecodes as they are compiled. This is done through the use of a JIT compiler that runs on a Java Virtual Machine, which in turn runs on any type of computer system that supports Java. The entire disclosure of Lueh is related to optimization of computer code, which also must be implemented as computer code for the machine to be able to properly interpret the method. Therefore, the discussion of claims 9-10 form the basis for rejection of the present claims as well.

## *Conclusion*

6. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. The additional cited references pertain to the analysis of currently executing Java code and the "Just-In-Time" compilation performed therein. Various characteristics of JIT compilers are disclosed that relate to the optimization of code at runtime.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Syed J Ali whose telephone number is (703) 305-8106. The examiner can normally be reached on Mon-Fri 8-5:30, 2nd Friday off.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, William A Grant can be reached on (703) 308-1108. The fax phone number for the organization where this application or proceeding is assigned is (703) 872-9306.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-3900.

[illegible]

Syed Ali
November 12, 2003

MAJID A. BANANKHAH
PRIMARY EXAMINER